Design Document for P1 HTTP Proxy

Partner A: Reese Stewart

Partner B: Matthew Jagen

1. Socket Management

Describe 1) how many socket objects are used, 2) the variable name for each socket, and 3) the purpose of each socket object.

We always have at least 3 sockets, one for the proxy to establish client connections (***proxy***), one to communicate between client and proxy (***connection.client***), and another to communicate between the server and proxy (***connection.server***).

Those last two sockets are unique per connection.

2. Multi-threading

Overall Design

Describe how multithreading is used in your program. You must include 1) the number of threads you use including the main thread, 2) what each thread is for and doing, 3) any loops you have for each thread.
For our design, we decided to use pyuv event-based programming instead of multithreading. The pyuv loop is set up so that all sockets run inside the singular loop and all call on_read after receiving data. After arriving at on_read, the function determines if the socket was a client or server socket and calls client_read() or server_read(). The other pyuv handles that we use are a TTY handle to detect EOF on the terminal, a signal handle for detecting signal.SIGINT (keyboard interrupt), and the proxy TCP handle that establishes all connections from clients to the proxy.

Justification

Justify why your design is efficient and show evidence it can handle multiple client requests efficiently. Specify your testing scenario (how many requests were made, which websites were being used, etc
We are confident that our design is efficient and can handle multiple requests simultaneously. We tested our proxy using firefox and loading data-rich websites such as google, CNN, and youtube. The last of which makes many simultaneous requests to ad providers and various google services and it not only loaded quickly but also handled HD video streaming with ease.

3. Streaming

Describe how streaming is implemented in your Proxy and the parameter (i.e. num of bytes) used for streaming. Justify the choice of your parameter.
Streaming is done in our proxy by directly forwarding all data received by the TCP handles to the destination without waiting on anything other than the header. We ensure we have buffered/received the entirety of the header (it is buffered in a bytes object belonging to our connectionData class) before editing and streaming it. The pyuv TCP handle does not specify how many bytes are used for streaming so our code is designed around detecting whether we have received the HTTP header terminator yet and buffering what we have if not, and modifying and then sending the header if we have. Once the header has been sent (tracked in a header_sent bool in our connectionData class), the TCP handles directly forward all data received.

4. Data structures

In the cell below, list any notable data structures you used and justify the use of the data structure. Also specify how you handle synchronization for the data structure if there were any need for synchronization. If none, you can say "None".

connectionData: This is the core data structure for this project. This object stores the client socket, server socket, log dictionary (stores each of the relevant fields for creating a json file), header and response buffers (as data is streamed in, we place them into the relevant buffers so we can send the entire header/response), host name (just a string), and a couple of booleans to track if the header has been sent, if the request was CONNECT, and if the response has been logged. These were then stored in our connections list, which allowed us access to any of the relevant information for communication with the client/server.

Connections: This is just a global list of connectionData objects which we search any time that our proxy receives some data, allowing us to determine if we have a client or server socket, and all the important information stored within the connectionData object

5. How shutdown is handled

Describe how you handled the shutdown gracefully.
Shutdown is handled by iterating through all connectionData objects in our connections list and closing the client and server TCP handles. After closing the handles, it logs any requests that have not yet been logged and finally stops the pyuv loop. Because we're using pyuv we did not have to worry about daemon threads or synchronization while freeing resources. There is an edge case where our keyboard interrupt could occur while a client or server TCP handle is closed or nonexistent, but we cover it by checking before we call close().

6. Error handling

Describe how you handle unexpected errors by specifying 1) what kind of errors you may encounter 2) what you do for each error.

Unresolved domain name: If for whatever reason the domain name cannot be resolved when we call ***gethostname(hostname)***, we just print out ”unable to resolve domain name”

UTF decoding errors: We could get some errors while decoding the bytes received from server/client, so to handle these, we just used ***.decode('utf-8','backslashreplace')***

When creating the json logs: We use ***with open*** which allows us to create a new json, write to it, and then close the file upon completion.

7. Any libraries used

List any libraries that you used for the implementation and justify the usage.

We used pyuv to make our proxy run multiple clients simultaneously and efficiently. Running the proxy using an event-based library instead of multithreading allows us to easily relinquish resources when they are no longer needed and allows us to not worry about synchronization. The only downsides with it are that it is a bit harder to tie together client and server sockets for each client connection and coordinate connection-specific logic across different event callbacks.

8. Reflection

What was the most challenging part working on this project? Most fun part?
The most challenging part of the project was knowing where to get started or how we should structure our code. The most fun part of the project is finishing it and messing around with our functional proxy.

If you are to do this all over again, how would you do it differently?

We would definitely benefit from having more time, so it would have been better if we started earlier, especially because of the debugging issues that we ran into.

Reflection on pair programming

Log of the amount of time spent driving and the amount of time spent working individually for each part (e.g., X drives 1 hour; Y drives 45 minutes; X works alone for 1 hour, etc.)

- *For Check point:*
    - *Reese drives: 0.5hr*
    - *Matthew drives: 0.5hr*
    - *Reese drives: 0.5hr*
    - *Matthew drives: 0.5hr*
    - *Reese drives: 0.5hr*
    - *Matthew drives: 1hr*
    - *Reese drives: 1hr*
    - *Matthew drives: 1hr*
    - *Reese drives: 0.5hr*
    - *Matthew drives: 0.5hr*
- *For Final submission:*
    - *Matthew drives: 0.5hr*
    - *Reese drives: 0.5hr*
    - *Matthew drives: 0.5hr*
    - *Reese drives: 0.5hr*

- *Matthew drives: 0.5hr*

What went well/or not-so-well doing pair programming? What was your take away in this process?

We worked pretty well together and we had really good communication with each other. Overall it was a solid experience which isn't necessarily guaranteed with having a random partner to work with.

Submission Instruction

Remember to export to pdf and push it to your github team repo under the project root